Modular Goniometer Controller type MGC5 – principle of operation

Version: 1.10
Date: 16 January 2003
File: mgc_operation_1_10.doc

Time units

The time base for the controller is set in mgc_comm.h file:
CLOCK = 20 * 2^20 Hz = 20971520 Hz – CPU clock
TPUCLK = 5 * 2^20 Hz = 5242880 Hz - TPU clock (TCR1)

The time unit for motor movement is set as:
1/TPUCLK = 1/(5*(2^20) ≈ 190,73486328125 ns
The time unit is used in Per_x registers, it means the period of microstep is quantified in 1/TPUCLK.

The time unit for the measurements with a point detector is set in mgc_comm.h file:
CORR_PER = (TPUCLK+50)/100 - integer constant for TPU (CORR_PER=52429), appropriated for periodic correction a point detector non-linearity. The correction is calculated in a constant period, which is equal roughly 10ms (10.00003814697 ms). SlicePer is quantified in ~10ms unit. Correction parameter is DeadTime represented in 1ns, written into the EEPROM in service mode.

Basic registers

There are plenty registers used as a controller status, the axis and slit positions, and mode of operation and commands.

Information on the state of the controller is available in status register GlobStat:

Table 1. GlobStat register - status of the controller

Item	Name	code	description
1	StatStart	0	start conditions, just after hardware reset
2	StatSync	1	when at least one axis is not synchronized
3	StatReady	2	all axes are synchronized and ready to move
4	StatBusy	3	controller is executing command
5	StatQuit	4	state after sleep request
6	StatService	5	service state

Register Events contains additional information:

Table 2. Events register – bit fields

Item	Name	bit	description
1	RotErr0	0	Error encountered in motor 1 movement
2	RotErr1	1	Error encountered in motor 2 movement
3	RotErr2	2	Error encountered in motor 3 movement
4	RotErr3	3	Error encountered in motor 4 movement
5	RotErr4	4	Error encountered in motor 5 movement
6	ManSync	13	Synchronization forced manually
7	PFail	14	Power fail action performed
8	EmStop	15	Emergency stop button was pressed

Available information on the motor state:

Table 3. Motors state

Item	Name	#bit	Type	Description
1	Sync	4..0	Int	bit field, 1 – revolution slit synchronized
2	Sync	12..8	Int	bit field, 1 - home slit synchronized
3	Start	12..8	Int	bit field, 1 – movement parameter error
4	PosHi_x: PosLo_x		Long	current axis position [microstep]
5	InPos	4..0	Int	bit field, 1 - in position, 0 – in motion
6	RevLeftHi_x: RevLeftLo_x		Long	left edge of x axis revolution slit [microstep]
7	RevRightHi_x: RevRightLo_x		Long	right edge of x axis revolution slit [microstep]
8	HomOnHi_x: HomOnLo_x		Long	beginning of x axis home slit [microstep]
9	HomOfHi_x: HomOfLo_x		Long	end of x axis home slit [microstep]

For revolution slits, edge positions of the slit are shown as left and right. For home slits, edge positions of the slit are exposed as falling (on, into the slit) and rising (off, from the slit).

Motor control – motion parameters. The registers shown below (Table 4. Motion parameters set) control each movement.

Table 4. Motion parameters set

Item	Name	#bit	type	description
1	MotionChk	4..0	int	bit field, 1 – motion check on, 0 - off
2	Operation		int	type of operation according to **Błąd! Nie można odnaleźć źródła odwołania.**
3	TarHi_x: TarLo_x		long	target position for x axis [microsteps]
4	PerHi_x: PerLo_x		long	half period of microstep for x axis [1/TPUCLK]
5	Start	4..0	int	bit field, 1 on bit position starts process
6	Stop	4..0	int	bit field, 1 on bit position stops process

Note: The half period is set with 13-bit resolution on the most significant bits. Maximum value of the half period is 20-bit, so the value may be truncated on the lowest bits.

The controller can work in several modes by writing mode of operation to register Operation.

Table 5. Operation types

Item	Name	code	description
1	GoTo	0	simple move operation
2	CcdSmo	1	movable scan with CCD detector
3	CcdSmi	2	static scan with CCD detector
4	CntSmo	3	movable scan with point detector
5	CntStep	4	static scan with point detector
6	DoSyncRev	5	revolution synchronization
7	DoSyncHome	6	home synchronization
8	Shutter	7	independent shutter control
9	MWSleep	8	standby
10	WakeUp	9	wake up from standby
11	AutoSlit	10	slit correction measurement
12	DoInitMot	11	motors initialisation
13	EraseEep	12	erase EEPROM

14	CntTime	13	limited counting to time or counts value

Procedures

GoTo procedure

There are two types of movement: normal and ramp. The controller chooses type of movement itself upon the value of velocity:
- for period less than THR_HPER, the motors accelerate and decelerate on the ramp,
- for period equal or higher than THR_HPER, the motors move with constant speed.

All motors are independent – it is possible to start any number of motors in the same time. All motors start simultaneously. Starting sequence is as follow:
- operation, target and period must be set prior,
- write bit mask into Start register,
- request is acknowledged by zeroing appropriate bits in Start register.

If the target is out of range, flag Error is set on corresponding bit. When the motor starts, appropriate bit is cleared in InPos register, which is zero as long as the motor is running. Any motor movement is signalling by global signal MOTORS# and LED named MOTORS shining.
There are defined the scanning directions for each axes individually. If movement is in opposite direction, procedure removing backlash (gear clearance) is applied:
- the target position is enhanced by GEAR_DIST,
- after achieving the target position, motor goes back to target minus GEAR_DIST with speed GEAR_HPER.

Each axis can be stopped by writing to Stop register bit mask. Request is acknowledged by zeroing appropriate bit in Stop register. The motor is stopped instantly or gradually depending on speed. In the case stopping motor by Stop, procedure removing gear clearance is not launched. There are current positions in Position registers. The InPos flags signal achieving target position for each axes separately.
If bit in MotionChk register is set, the movement is checked in two ways:
- the revolution slit is tested, distance between hypothetical centre of the slit and real centre of the slit must be lower than a predefined number – a margin of tolerance,
- the distance between last slit position and current position must be lower than the predefined number

Otherwise the procedure stopping the motors is invoked, the motors stop immediately or with braking on the ramp. A gear clearance is not removed. Appropriate SYNCR_x and SYNCH_x flags are cleared.

CcdSmo procedure

The procedure is used for scans with CCD detector. This procedure uses GoTo type movement, i.e. target and period must be set. After Start setting, the controller is waiting for the internal signal SHOPEN# low from the shutter controller. If SHOPEN# signal is recognised, the shutter is open, then the motors start. When the motors stop, the controller is waiting for the signal SHOPEN# high, then the procedure is finished. The procedure uses implicitly backslash correction. The scans are possible in backslash free direction only. Maximum speed should not exceed the THR_HPER value due to accelerates and decelerates at the beginning and the end of the movement.

CcdSmi procedure

The procedure is static – any motor doesn't move. The controller is waiting for SHOPEN# high, then the procedure is finished. This procedure is not necessary in a single thread operation.

CntSmo procedure

CntSmo procedure is used for measurement of a reflection in motion, using the point detector. This procedure is able to create a profile of the reflection. The reflection may be divided into the segments (slices). Maximum number of segments is 1000.
In order to initiate the procedure, following parameters must be set:

- SliceLen (long) – length (width) of the slice in microsteps, SliceLen > 0,
- ScanAxis (int) – reference axis of the scan, axes are numbered from 1 to 5,
- Tar_x – target for reference axis,
- Per_x – period for reference axis, must be higher then THR_HPER,
- Start – bit field for appropriate axes,
- optionally - movement of the remaining axes can be invoked by Tar_x, Per_x and bits in Start register

Last slice of the profile may be shorter than SliceLen depending on the values of Tar_x and SliceLen. This is recommended to keep:

Tar_x = N * SliceLen where N = 1,2,3,...,1000.

Total time of the measurement is equal:

Time_tot = abs(Tar_x – Pos_x)* Per_x where x is reference axis

Total time of the scan is equal time of the longest movement (when other axes are active). There are no restrictions on the Tar_x and Per_x, even on the scan direction.
NOTE:
Meaningful scans are only performed in backlash free direction. The Time_tot of all moving axes should be identical to produce meaningful scan data.
When the procedure is starting, signal SHRQMOT# is activated, then the controller is waiting on signal SHOPEN#. The motors start and the counter becomes active. Impulses coming from the point detector are counted. Results are immediately available on the MODBUS registers. Index of the last completely measured slice of the profile is available in CurrProf register. The counts are corrected in each ~10ms step, the correction depends on DeadTime. Each ~10ms sample which belongs to two adjacent slices is divided using linear interpolation, parts of the sample are added to slices. At the end of the movements, counting is stopped, signals MOTORS# and SHRQMOT# are switched off, and the controller is waiting for the rising signal SHOPEN#.

CntStep procedure

CntStep procedure is used for static measure a reflection using a point detector. Profile of reflection is built by sequence: measure, move, measure, move, ... ,move, measure.
The movements between points of the profile are done with default parameters. Following parameters are necessary for this procedure:
- Tar_x – for first step only, length of the step is calculated upon the difference Tar_x and Pos_x,
- SlicePer – the time of the measurement of one element of the profile in ~10ms
- Slices – number of movements in profile, number of measurement of Slices+1

Whole distance is calculated as (Slices * SliceLen), where SliceLen is abs(Tar_x – Pos_x). The controller starts the sequence by request shutter opening (SHRQMOT# signal) to the shutter controller. When SHOPEN# signal turns on, the counter is active in SlicePer time. After this time, the motors move and stop, then next slice of the profile is measured. After the last slice of the profile, shutter request (SHRQMOT#) signal is taken off, the controller is waiting for shutter closing (SHOPEN#). The shutter is open during whole procedure. Pos_x register is updated after each step.

CntTime procedure

The procedure is static - no movement is applied. The shutter is opened by SHRQMOT# signal, when the shutter is open, SHOPEN# signal initiates time counting. The value of the elapsed time is available in CTime register in ~10ms units. Impulses coming from the point detector are counted and corrected due to the dead time. The number of counts is available in CCount register. The values in CTime and CCount registers are consistent. Stop conditions arise when:
- the time is equal value of MaxCTime register,
- or when the counts are equal or higher the number in MaxCCount register

At the stop conditions, the SHRQMOT# signal goes high and the controller waits for rising SHOPEN# signal.
Writing into Start register zeros CTime and CCount registers. The registers are modified every ~10ms. The last acquired number of counts is available in CStat register. The value is not corrected due to the dead time.

The correction is made as follow:

```
if(6 * counts < GvTotRdt)           // magic condition
{
    if(counts != 0) counts = MAX_PLUS/(MAX_PLUS/counts - MAX_PLUS/GvTotRdt);
}
else
{
    counts = (12 * counts + 5)/10;  // saturation @ 120%
}
```

where:
- counts – current number of counts,
- GvTotRdt is the number corresponding to dead time,
- MAX_PLUS is equal 0x7FFFFFFF

Shutter operation

This mode is intended for direct shutter control. The shutter is opened by writing ones into Start register, and closed by writing ones into Stop register.

DoSyncRev operation

DoSyncRev operation is used to finding revolution slit and setting internal counter to proper value (modulo 12800). After this procedure, position of motors is precisely known in one revolution range.
The procedure has defaults parameters. Speed is limited to SEEK_HPER. The procedure is invoked by writing ones into Start register. Appropriate bits in Sync register are cleared, then the motors move according ScanDir in limited range to one revolution. After synchronization, the revolution slit on the axis is placed into the midpoint of the photointerrupter. Slit correction from the EEPROM is applied. Pos_x register is modified to modulo 12800 value, flags SYNCR_x, InPos_x are set. On the error condition (missing slit), ERR_x bit is set. Correction Slit_x means distance from slit centre to closest full-step point. The range of Slit_x is (-128, 127).
“Manual synchronization” is implemented in order to make some particular tests by simple setting appropriate bits on positions 0...4 in Sync register (1017). After writing into Sync register, flag ManSync in register Event is set.

DoSyncHome procedure

DoSyncHome procedure takes effect only with the motors having bit SYNCR_x set.
The procedure starts after writing ones into Start register of choose axes. No movement is applied. If:
- axis is on Home slit,
- value Pos_x is 0 modulo 12800,
- flag SYNCR_x is set,

register Pos_x is zeroed and flag SYNCH_x is set, otherwise bit ERR_x is set. If all flags SYNCR and SYNCH are set, the controller goes in StatReady state, otherwise it remains in StatSync state.
“Manual synchronization” is implemented in order to make some particular tests by simple setting appropriate bits on positions 5...9 in Sync register (1017). After writing into Sync register, flag ManSync in register Event is set.

MWSleep procedure

The MWSleep procedure is intended to put the controller into standby mode. At the beginning of this state the approximate positions of axes are written into EEPROM, then controller is ready to turn off. The motors are switched off. All commands are disabled except WakeUp command.

Notes:

1. Before going in MWSleep state, it is mostly recommended to move all axes on home position using GoTo command with Tar_x=0.
2. Controller goes in MWSleep state at PowerFail condition.

WakeUp procedure

The controller is reset by hardware like by power-on.

AutoSlit procedure

It is useful procedure for finding slit corrections for axes. Calling the AutoSlit is possible in service mode only. All axes are desynchronized. The controller looks for the revolution slit, then calculates distance from full-step position to the slit centre. This distance is a slit correction called shortly “slit”. It appears in Slit_x registers. This procedure should be used only by service personnel.

DoInitMotor procedure

DoInitMotor procedure desynchronizes all axes, resets motor drivers – motors move on full-step positions. From Pos_x register are subtracted Slit_x correction values. It may be used as a part of more complex initializing procedure.

EraseEep procedure

EraseEep procedure is available in service mode only. It is useful in early initial stage; the procedure should not be used later.

Power-on sequence

StatStart

State StatStart of the controller is at the hardware initialisation. The several values from the EEPROM are read: serial number, dead time, scan directions, slit corrections, last position of axes at power-off. If the checksum of EEPROM contents is failed, no synchronization is available. If the checksum is right, last approximate position (1-degree accuracy) is read and written into Pos_x registers, then Slit_x corrections are subtracted from these registers. Flags SYNCR_x and SYNCH_x are cleared. Tar_x registers are zeroed. The controller goes to StatSync.

StatSync

In StatSync state the controller is able to communicate using MODBUS protocol on a serial interface.
Master can rewrite a new position into Pos_x registers, move to Tar_x position, and make the revolution synchronization (DoSyncRev) or the home synchronization (DoSyncHome). If synchronization is successfully done, the controller goes into StatReady, otherwise the controller remains in StatSync state with partially set Sync register. The operation can be repeated. There is no way to achieve StatReady without full synchronization.

StatReady

In this state all types of scan and movement are available. It is possible to move from StatReady to StatService state.

StatBusy

It is intended to indication that operation is running, writing into MODBUS registers is ignored, fixed as error(writing to Start) or executed in case writing to Stop register

StatQuit

The StatQuit is after executing MWSleep command. The controller is frozen and waits for power-off or WakeUp command.

StatService

The service mode is prepared in order to set and change the crucial parameters. The following parameters can be changed:
- serial number SvSerialNo,
- two special words Memo0 and Memo1,
- scan directions SvScanDir,
- parameter of counter correction SvDeadTime,
- slit corrections Slit_1,..,Slit_5

The following operations can be executed:
- finding correction for slits – AutoSlit operation,
- EEPROM erasing.

To enter in service mode, the following sequence is necessary:
- write MAGIC number into SvPasswd,
- read all service registers (from SvPasswd to Slit_5 inclusive),
- calculate and write into SvPasswd checksum modulo 0x10000,
- read GlobStat register to ensure the service mode is entered,
- do related to service mode operations,
- exit from service mode by writing MAGIC number into SvPasswd

Note:
After exiting the service mode, the axis synchronization is required.
There is no automatic write of the measured Slit_x into the EEPROM.
The synchronization made after AutoSlit operation uses the new Slit_x.

Emergency stop procedure

The emergency stop procedure is invoked by pressing the stop/safety button on the front panel of the controller. The following steps are applied:
- the motors are switched off,
- the Events register is set to 0x8000,
- all Sync flags are cleared,
- the motor drivers are disabled

The emergency state can be quitted when:
- all axes are in position,
- the stop button is released

Writing 0x8000 into Events register quits the emergency state. The controller is like after reset state.

MODBUS registers summary

Users registers

address	name	description
1001	FwareVer	firmware version
1002	Memo0	memory 0 word
1003	Memo1	memory 1 word
1004	SerialNo	serial number
1005	DeadTime	dead time parameter
1006	ScanDir	scan directions
1007	Slit_1	slit correction for axis no 1
1008	Slit_2	slit correction for axis no 2
1009	Slit_3	slit correction for axis no 3
1010	Slit_4	slit correction for axis no 4
1011	Slit_5	slit correction for axis no 5
1012	IntPort	input port – sources of the interrupt
1013	InPort	Inputs
1014	OutPort	Outputs
1015	GlobStat	Status
1016	Events	events: emergency stop, power fail, manual sync, synchronization fail
1017	Sync	synchronization flags
1018	InPos	in position flags
1019	PosHi_1	position of axis 1, high word
1020	PosLo_1	position of axis 1, low word
1021	PosHi_2	
1022	PosLo_2	
1023	PosHi_3	
1024	PosLo_3	
1025	PosHi_4	
1026	PosLo_4	
1027	PosHi_5	
1028	PosLo_5	
1029	RevLeftHi_1	left edge of revolution slit #1, high word
1030	RevLeftLo_1	left edge of revolution slit #1, low word
1031	RevRightHi_1	right edge of revolution slit #1, high word
1032	RevRightLo_1	right edge of revolution slit #1, low word
1033	RevLeftHi_2	
1034	RevLeftLo_2	
1035	RevRightHi_2	
1036	RevRightLo_2	
1037	RevLeftHi_3	
1038	RevLeftLo_3	
1039	RevRightHi_3	
1040	RevRightLo_3	
1041	RevLeftHi_4	
1042	RevLeftLo_4	
1043	RevRightHi_4	
1044	RevRightLo_4	
1045	RevLeftHi_5	
1046	RevLeftLo_5	
1047	RevRightHi_5	
1048	RevRightLo_5	
1049	HomOnHi_1	position at high-to-low transition on home slit #1, high word
1050	HomOnLo_1	position at high-to-low transition on home slit #1, low word
1051	HomOfHi_1	position at low-to-high transition on home slit #1, high word

1052	HomOfLo_1	position at low-to-high transition on home slit #1, low word
1053	HomOnHi_2	
1054	HomOnLo_2	
1055	HomOfHi_2	
1056	HomOfLo_2	
1057	HomOnHi_3	
1058	HomOnLo_3	
1059	HomOfHi_3	
1060	HomOfLo_3	
1061	HomOnHi_4	
1062	HomOnLo_4	
1063	HomOfHi_4	
1064	HomOfLo_4	
1065	HomOnHi_5	
1066	HomOnLo_5	
1067	HomOfHi_5	
1068	HomOfLo_5	
1069	MotionChk	motion check flags
1070	Operation	type of operation
1071	TarHi_1	target for axis #1, high word
1072	TarLo_1	target for axis #1, low word
1073	PerHi_1	speed of axis #1, high word
1074	PerLo_1	speed of axis #1, low word
1075	TarHi_2	
1076	TarLo_2	
1077	PerHi_2	
1078	PerLo_2	
1079	TarHi_3	
1080	TarLo_3	
1081	PerHi_3	
1082	PerLo_3	
1083	TarHi_4	
1084	TarLo_4	
1085	PerHi_4	
1086	PerLo_4	
1087	TarHi_5	
1088	TarLo_5	
1089	PerHi_5	
1090	PerLo_5	
1091	Start	start bit field
1092	Stop	stop bit field
1093	VGain	gain of the amplifier
1094	HIRef	high level of window comparator
1095	LIRef	low level of window comparator
1096	HvRef	high voltage control
1097	MaxCTimeHi	maximum time of scan, high word
1098	MaxCTimeLo	maximum time of scan, low word
1099	MaxCCountHi	maximum number of counts, high word
1100	MaxCCountLo	maximum number of counts, low word
1101	Slices	number of slices of profile
1102	SlicePer	time of slice in ~10ms
1103	ScanAxis	reference axis for cntsmo type scan
1104	SliceLenHi	length of slice in microsteps, high word
1105	SliceLenLo	length of slice in microsteps, low word
1106	CTimeHi	elapsed time of scan, high word
1107	CTimeLo	elapsed time of scan, low word
1108	CCountHi	entire counts number, high word
1109	CCountLo	entire counts number, low word

1110	CStat	not corrected counts in ~10ms
1111	CurrProf	currently measured slice of profile
1112	ProfHi_1	slice #1 of profile, high word
1113	ProfLo_1	slice #1 of profile, low word
...	...	...
3112	ProfHi_1000	slice #1000 of profile, high word
3113	ProfLo_1000	slice #1000 of profile, low word

Service registers

5001	SvPasswd	password register
5002	SvMemo0	extra memory #0
5003	SvMemo1	extra memory #1
5004	SvSerialNo	serial number
5005	SvDeadTime	dead time parameter
5006	SvScanDir	scan direction
5007	SvSlit_1	slit correction for axis #1
5008	SvSlit_2	
5009	SvSlit_3	
5010	SvSlit_4	
5011	SvSlit_5	

Appendix A

```
/*********************************************************************
*
* File:         mgc_comm.h
*
* Description:  MGC common configuration definitions
*
* Created by:   Marek Wnuk
*
* History:
*--------------------------------------------------------------------
* $Log: $
* M.WNUK (C) 2001
*
*********************************************************************/

#ifndef _MGC_COMMON_
#define _MGC_COMMON_
#define AX_NBR      5             // Number of axes in the manipulator
#define AX_MASK     (0x1f>>(5-AX_NBR))
#define INV_POS     0x80000000    // Invalid position marker
#define INV_REV     0x8000        // Invalid revolution marker
#define NUL_POS     0             // Initial (HOME) position
#define CLOCK       20971520L     // 20 * 2^20 Hz CPU time base
#define TPUCLK      (CLOCK/4)     //  5 * 2^20 Hz TPU time base
#define CORR_PER    ((TPUCLK+50)/100) // count correction period (in TPU ticks)
#define Baud        19200
#define SLADDR      1        // MODBUS slave address
#define MAGIC       0x4d57   // Checksum and password constant
#define THR_HPER    128      // Ramp/normal move threshold halfperiod (in TPU ticks)
#define GEAR_DIST   1000     // Gear adjustment distance

// The following macros limit the number of registers read/written in one
// transaction:

#define FC03_MAX    125      // Max. number of read registers
#define FC16_MAX    125      // Max. number of written registers

/* MGC EEPROM data structure */

typedef struct{
    int Memo0;          // The 1st EEPROM word (reserved)
    int Memo1;          // The 2nd EEPROM word (reserved)
    int SerialNo;       // Serial number
    int DeadTime;       // Counter dead time [ns]
    int ScanDir;        // Scan direction
    int Slit[5];        // Slit correction
    int PwrFailRev[5];  // Rough powerfail position [revolutions]
    int ChkSum;         // EEPROM checksum
} MGC_EEP;

// In Modicon Standard, from the user point of view, the registers are indexed
// from 1 to 65536, nevertheless, inside the frames the indices are decremented
// by one to fit 16-bit word (e.g. register number 1001, as seen in Calta's Mdbus
// application, results in index 1000 in the corresponding frame and vice versa).

/* MGC Modbus NORMAL registers */

// In MGC, NORMAL registers are accessible via MbInd(n), where n is substituted from
// MB_REGS enumeration and INT_BASE stands for the index of the very first one
// (e.g. MbInd(FwareVer) gives 1000, which is o.k. for the frame; however, one has to
// add 1 to it to reflect user convention -  1001 is the right number for Mdbus/Calta).

// MbInd(n) defines Modbus register index for READ_N_REGISTERS and WRITE_N_REGISTERS
// functions inside the communication frames, and must be incremented by one to
// reflect the external (user) standard.
```

```
#define PROF_LEN    1000                  // Profile table size (in long words)
#define INT_NBR     (ProfHi_1+2*PROF_LEN) // MGC holding (NORMAL) registers count
#define INT_BASE    1000                  // NORMAL registers base address
#define INT_MAX     INT_BASE+INT_NBR      // Max register address (for NORMAL registers)
#define MbInd(n)    ((n)+INT_BASE)        // Selected NORMAL register index

typedef enum{

// A. Service (read only, for user modes).

    FwareVer,       // Firmware version
    Memo0,          // The 1st EEPROM word (reserved)
    Memo1,          // The 2nd EEPROM word (reserved)
    SerialNo,       // Serial number
    DeadTime,       // Counter dead time [ns]
    ScanDir,        //      7       6       5       4       3       2       1       0
                    //     --      --      --   SDIR4   SDIR3   SDIR2   SDIR1   SDIR0
    Slit_1,         // Slit correction (ax. #1)
    Slit_2,         // Slit correction (ax. #2)
    Slit_3,         // Slit correction (ax. #3)
    Slit_4,         // Slit correction (ax. #4)
    Slit_5,         // Slit correction (ax. #5)

// B. General Status.

    IntPort,        //     15      14      13      12      11      10       9       8
                    //     PF     MF4     MF3     MF2     MF1     MF0   HOME4   HOME3
                    //
                    //      7       6       5       4       3       2       1       0
                    //  HOME2   HOME1   HOME0  EHOME4  EHOME3  EHOME2  EHOME1  EHOME0

    InPort,         //     15      14      13      12      11      10       9       8
                    //      0  SHACK_   IFIP3   IFIP2 SHOPEN_   COLL_ RSTSFTY_  SHSEL_
                    //
                    //      7       6       5       4       3       2       1       0
                    //     K3      K2      K1      K0    GPI3    GPI2    GPI1    GPI0

    OutPort,        //  only bits 7..4 - GPO - are writeable
                    //     15      14      13      12      11      10       9       8
                    //   DIR4    DIR3    DIR2    DIR1    DIR0     RDY_   FAIL_ MOTORS_
                    //
                    //      7       6       5       4       3       2       1       0
                    //   GPO3_  GPO2_   GPO1_   GPO0_   ENMOT RSTMOT_ SHRQMOT_  SHEN_

    GlobStat,       // Global state (read only) as in MGC_STAT
                    //

    Events,         // Controller status bit fields
                    //     15      14      13      12      11      10       9       8
                    // EMSTOP   PFAIL MANSYNC      --      --      --      --      --
                    //
                    //      7       6       5       4       3       2       1       0
                    //     --      --      -- ROTERR4 ROTERR3 ROTERR2 ROTERR1 ROTERR0

// C. Motor Status.

    Sync,           //     15      14      13      12      11      10       9       8
                    //     --      --      --  HSYNC4  HSYNC3  HSYNC2  HSYNC1  HSYNC0

                    //      7       6       5       4       3       2       1       0
                    //     --      --      --  RSYNC4  RSYNC3  RSYNC2  RSYNC1  RSYNC0

    InPos,          //      7       6       5       4       3       2       1       0
                    //     --      --      --  INPOS4  INPOS3  INPOS2  INPOS1  INPOS0

    PosHi_1,        // MSW of motor #1 position
    PosLo_1,        // LSW of motor #1 position
```

```
    PosHi_2,        // MSW of motor #2 position
    PosLo_2,        // LSW of motor #2 position

    PosHi_3,        // MSW of motor #3 position
    PosLo_3,        // LSW of motor #3 position

    PosHi_4,        // MSW of motor #4 position
    PosLo_4,        // LSW of motor #4 position

    PosHi_5,        // MSW of motor #5 position
    PosLo_5,        // LSW of motor #5 position


    RevLeftHi_1,    // Rev. slit start position (MSW)
    RevLeftLo_1,    // Rev. slit start position (LSW)
    RevRightHi_1,   // Rev. slit end position   (MSW)
    RevRightLo_1,   // Rev. slit end position   (LSW)

    RevLeftHi_2,    // Rev. slit start position (MSW)
    RevLeftLo_2,    // Rev. slit start position (LSW)
    RevRightHi_2,   // Rev. slit end position   (MSW)
    RevRightLo_2,   // Rev. slit end position   (LSW)

    RevLeftHi_3,    // Rev. slit start position (MSW)
    RevLeftLo_3,    // Rev. slit start position (LSW)
    RevRightHi_3,   // Rev. slit end position   (MSW)
    RevRightLo_3,   // Rev. slit end position   (LSW)

    RevLeftHi_4,    // Rev. slit start position (MSW)
    RevLeftLo_4,    // Rev. slit start position (LSW)
    RevRightHi_4,   // Rev. slit end position   (MSW)
    RevRightLo_4,   // Rev. slit end position   (LSW)

    RevLeftHi_5,    // Rev. slit start position (MSW)
    RevLeftLo_5,    // Rev. slit start position (LSW)
    RevRightHi_5,   // Rev. slit end position   (MSW)
    RevRightLo_5,   // Rev. slit end position   (LSW)

    HomOnHi_1,      // Home slit start position (MSW)
    HomOnLo_1,      // Home slit start position (LSW)
    HomOfHi_1,      // Home slit end position   (MSW)
    HomOfLo_1,      // Home slit end position   (LSW)

    HomOnHi_2,      // Home slit start position (MSW)
    HomOnLo_2,      // Home slit start position (LSW)
    HomOfHi_2,      // Home slit end position   (MSW)
    HomOfLo_2,      // Home slit end position   (LSW)

    HomOnHi_3,      // Home slit start position (MSW)
    HomOnLo_3,      // Home slit start position (LSW)
    HomOfHi_3,      // Home slit end position   (MSW)
    HomOfLo_3,      // Home slit end position   (LSW)

    HomOnHi_4,      // Home slit start position (MSW)
    HomOnLo_4,      // Home slit start position (LSW)
    HomOfHi_4,      // Home slit end position   (MSW)
    HomOfLo_4,      // Home slit end position   (LSW)

    HomOnHi_5,      // Home slit start position (MSW)
    HomOnLo_5,      // Home slit start position (LSW)
    HomOfHi_5,      // Home slit end position   (MSW)
    HomOfLo_5,      // Home slit end position   (LSW)


// D. Motor Control.
```

```
    MotionChk,      //      7       6       5       4       3       2       1       0
                    //     --      --      --   MCHK4   MCHK3   MCHK2   MCHK1   MCHK0


    Operation,      // Scan mode or park request as in MGC_OPER

    TarHi_1,        // MSW of motor #1 target position
    TarLo_1,        // LSW of motor #1 target position
    PerHi_1,        // Requested motor #1 step period
    PerLo_1,        // Requested motor #1 step period

    TarHi_2,        // MSW of motor #2 target position
    TarLo_2,        // LSW of motor #2 target position
    PerHi_2,        // Requested motor #2 step period
    PerLo_2,        // Requested motor #2 step period

    TarHi_3,        // MSW of motor #3 target position
    TarLo_3,        // LSW of motor #3 target position
    PerHi_3,        // Requested motor #3 step period
    PerLo_3,        // Requested motor #3 step period

    TarHi_4,        // MSW of motor #4 target position
    TarLo_4,        // LSW of motor #4 target position
    PerHi_4,        // Requested motor #4 step period
    PerLo_4,        // Requested motor #4 step period

    TarHi_5,        // MSW of motor #5 target position
    TarLo_5,        // LSW of motor #5 target position
    PerHi_5,        // Requested motor #5 step period
    PerLo_5,        // Requested motor #5 step period

    Start,          //     15      14      13      12      11      10       9       8
                    //     --      --      --    ERR4    ERR3    ERR2    ERR1    ERR0
                    //      7       6       5       4       3       2       1       0
                    //     --      --      --  START4  START3  START2  START1  START0

    Stop,           //      7       6       5       4       3       2       1       0
                    //     --      --      --   STOP4   STOP3   STOP2   STOP1   STOP0


// E. DACs (rw).

    VGain,          // AIN gain control (U2/counter)
    HlRef,          // Upper threshold (INLEVELH)
    LlRef,          // Lower threshold (INLEVELL)
    HvRef,          // VPROG for M2/mgc_mb

// F. Counter Control (wo).

    MaxCTimeHi,     // MSW of CntTime time limit [10ms]
    MaxCTimeLo,     // LSW of CntTime time limit [10ms]

    MaxCCountHi,    // MSW of CntTime count limit
    MaxCCountLo,    // LSW of CntTime count limit

    Slices,         // Requested number of profile table entries
    SlicePer,       // Requested slice time in 10ms units

    ScanAxis,       // Index of the main scan axis [1 .. AX_NBR]
    SliceLenHi,     // Requested slice distance in microsteps (MSW)
    SliceLenLo,     // Requested slice distance in microsteps (LSW)


// G. Counter Results (ro).

    CTimeHi,        // MSW of current count time [10ms]
    CTimeLo,        // LSW of current count time [10ms]
```

```
    CCountHi,       // MSW of current pulse count
    CCountLo,       // LSW of current pulse count

    CStat,          // Current (raw) pulse density [pulses/10ms]

    CurrProf,       // Current completed profile slice count

    ProfHi_1,       // MSW of pulse count in 1-st time slice
    ProfLo_1        // LSW of pulse count in 1-st time slice

} MB_REGS;

// In the following aux. macros, the axes are numbered from 0 to AX_NBR-1
// for compatibility with standard arrays:

#define RevLeftHi(n)  (4*(n)+RevLeftHi_1)
#define RevLeftLo(n)  (4*(n)+RevLeftLo_1)
#define RevRightHi(n)  (4*(n)+RevRightHi_1)
#define RevRightLo(n)  (4*(n)+RevRightLo_1)
#define HomOnHi(n)  (4*(n)+HomOnHi_1)
#define HomOnLo(n)  (4*(n)+HomOnLo_1)
#define HomOfHi(n)  (4*(n)+HomOfHi_1)
#define HomOfLo(n)  (4*(n)+HomOfLo_1)

#define Slit(n)     ((n)+Slit_1)

#define PosHi(n)    (2*(n)+PosHi_1)
#define PosLo(n)    (2*(n)+PosLo_1)

#define TarHi(n)    (4*(n)+TarHi_1)
#define TarLo(n)    (4*(n)+TarLo_1)
#define PerHi(n)    (4*(n)+PerHi_1)
#define PerLo(n)    (4*(n)+PerLo_1)

#define ProfHi(n)   (2*(n)+ProfHi_1)
#define ProfLo(n)   (2*(n)+ProfLo_1)


/* MGC global states in GlobState (MB_REGS) */

typedef enum
{
    StatStart,          // Hardware Initialization
    StatSync,           // Home position seek
    StatReady,          // Ready for next operation
    StatBusy,           // Not ready (possible fault condition)
    StatQuit,           // Stop (position saved in EEPROM)
    StatService         // Special services (protected)
} MGC_STAT;


/* MGC operation requests in Request register (MB_REGS) */

typedef enum
{
    GoTo,               // Change position
    CcdSmo,             // CCD, scan in motion
    CcdSmi,             // CCD, scan in one position
    CntSmo,             // CNT, scan in motion
    CntStep,            // CNT, scan in n positions
    DoSyncRev,          // Synchronize selected axes within 1 revolution
    DoSyncHome,         // Synchronize selected axes in Home slit
    Shutter,            // Manual shutter control (start - stop)
    MWSleep,            // Save position and enter quit state
    WakeUp,             // Restart the controller
    DoAutoSlit,         // Find and set slit correction
    DoInitMot,          // Set all motors on magnetic positions
```

```
    EraseEep,           // Clear EEPROM (invalid checksum!)
    CntTime             // CNT, static timed scan
} MGC_OPER;


/* MGC Modbus SERVICE mode registers */


// In MGC, SERVICE registers are accessible via SvInd(n), where n is substituted from
// SV_REGS enumeration and SVC_BASE stands for the index of the very first one
// (e.g. MbInd(SvPasswd) gives 5000, which is o.k. for the frame; however, one has to
// add 1 to it to reflect user convention - 5001 is the right number for Mdbus/Calta).

// SvInd(n) defines SERVICE register index for READ_N_REGISTERS and WRITE_N_REGISTERS
// functions inside the communication frames, and must be incremented by one to
// reflect the external (user) standard.


#define SVC_NBR     (SvSlit_5+1)        // MGC holding (SERVICE) registers count
#define SVC_BASE    5000                // SERVICE registers base address
#define SVC_MAX     SVC_BASE+SVC_NBR    // Max register address (for SERVICE registers)

#define SvInd(n)    ((n)+SVC_BASE)      // Selected SERVICE register index

typedef enum{

    SvPasswd,       // Service mode request/password

    SvMemo0,        // The 1st EEPROM word (reserved)
    SvMemo1,        // The 2nd EEPROM word (reserved)

    SvSerialNo,     // Serial number

    SvDeadTime,     // Counter dead time [ns]

    SvScanDir,      //      7       6       5       4       3       2       1       0
                    //     --      --      --   SDIR4   SDIR3   SDIR2   SDIR1   SDIR0

    SvSlit_1,       // Slit correction (ax. #1)
    SvSlit_2,       // Slit correction (ax. #2)
    SvSlit_3,       // Slit correction (ax. #3)
    SvSlit_4,       // Slit correction (ax. #4)
    SvSlit_5        // Slit correction (ax. #5)

} SV_REGS;

// In the following aux. macros, the axes are numbered from 0 to AX_NBR-1
// for compatibility with standard arrays:

#define SvSlit(n)   ((n)+SvSlit_1)

#endif /* _MGC_COMMON_ */

/* end of file "mgc_comm.h" */
```